

PROJECT

Software Engineering Department



SOFTWARE CONSTRUCTION & DEVELOPMENT

Submitted by

ABDUL MOIZ | 65315

Lab Instructor

MAAM LAILA

Session Fall – 2023

BALOCHISTAN UNIVERSITY OF INFORMATION TECHNOLOGY, ENGINEERING AND
MANAGEMENT SCIENCES, QUETTA.

1. Introduction

1.1 Project Description

This project is a **Smart Inventory & Warehouse Management System**, developed using **Node.js, Express, MySQL, and React.js** as part of the Software Construction & Design (SCD) course.

The main goal of this project is to demonstrate **practical usage of Software Design Patterns** inside a real-world inventory system.

The system allows users to:

- Manage warehouse items
- Track stock levels
- Add new items
- Register suppliers
- View low-stock alerts
- Generate dashboard summaries

Design Patterns Used:

- **Factory Pattern** → For item creation
- **Composite Pattern** → For warehouse structure (Warehouse → Sections → Racks → Bins)
- **Decorator Pattern** → For adding extra metadata to items (Fragile, Heavy, Critical)
- **Observer Pattern** → For auto-trigger low-stock alerts
- **Facade Pattern** → For consolidated reporting (summary stats, alert counts)

This project fully implements Object-Oriented concepts and SCD design principles.

2. Design Pattern Usage

2.1 Factory Pattern

Problem It Solves:

System needs to create items with different configurations (id, name, sku, category) without repeating object construction code.

Implementation:

ItemFactory returns fully structured item objects simply by passing required fields.

Class-Level Structure:

- ItemFactory → Creates new items
- Returned Object → Contains id, sku, quantity, category, labels

UML (Text Format):

ItemFactory → ItemObject

2.2 Composite Pattern

Problem It Solves:

Warehouse structure has multiple nested levels:

Warehouse → Sections → Racks → Bins.

Managing them individually is difficult.

Implementation:

Composite classes: WarehouseGroup, WarehouseComponent, Section, Rack, Bin.

Class Structure:

```
WarehouseGroup
  |--- Section
  |   |--- Rack
  |   |   |--- Bin
```

2.3 Decorator Pattern

Problem It Solves:

Items need extra metadata like **Fragile**, **Critical**, **Heavy**, but we shouldn't create subclasses for every combination.

Implementation:

Decorators wrap items and add new metadata.

Classes:

- ItemDecorator (base)
- FragileDecorator
- CriticalDecorator
- HeavyDecorator

2.4 Observer Pattern

Problem It Solves:

When an item goes below minimum stock, system should automatically create alerts.

Implementation:

Observers watch inventory changes and trigger alerts when conditions match.

Class Structure:

- InventoryObserver
- AlertObserver
- Database table: alerts

2.5 Facade Pattern

Problem It Solves:

Dashboard needs combined statistics from multiple database tables.

Implementation:

ReportingFacade joins multiple MySQL queries and returns a single structured summary.

Facade Outputs:

- Total items

- Unique SKUs
- Low-stock items
- Supplier count
- Alerts by severity

3. Implementation Details

3.1 Technologies Used

Frontend:

- React.js + Functional Components
- Fetch API for backend communication
- CSS (Dark UI theme)

Backend:

- Node.js
- Express.js
- MySQL Database
- dotenv
- uuid
- CORS

Database:

Tables created:

- items
- suppliers
- supplier_items
- alerts

4. Screenshots / Output

1. Dashboard view

Smart Inventory & Warehouse System

SCD Project — Express + MySQL + React (Design Patterns)

Dashboard Items Add Item Suppliers Add Supplier Alerts

Refresh Data

Dashboard Summary

Total Items 4 Unique SKUs	Total Quantity 23 All items quantity	Low Stock Items 2 Below minimum stock	Suppliers 2 Registered suppliers
--	---	--	---

Alerts by Severity

Severity	Count
LOW_STOCK	2

Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts

Smart Inventory & Warehouse System

SCD Project — Express + MySQL + React (Design Patterns)

Dashboard Items Add Item Suppliers Add Supplier Alerts

Refresh Data

All Items

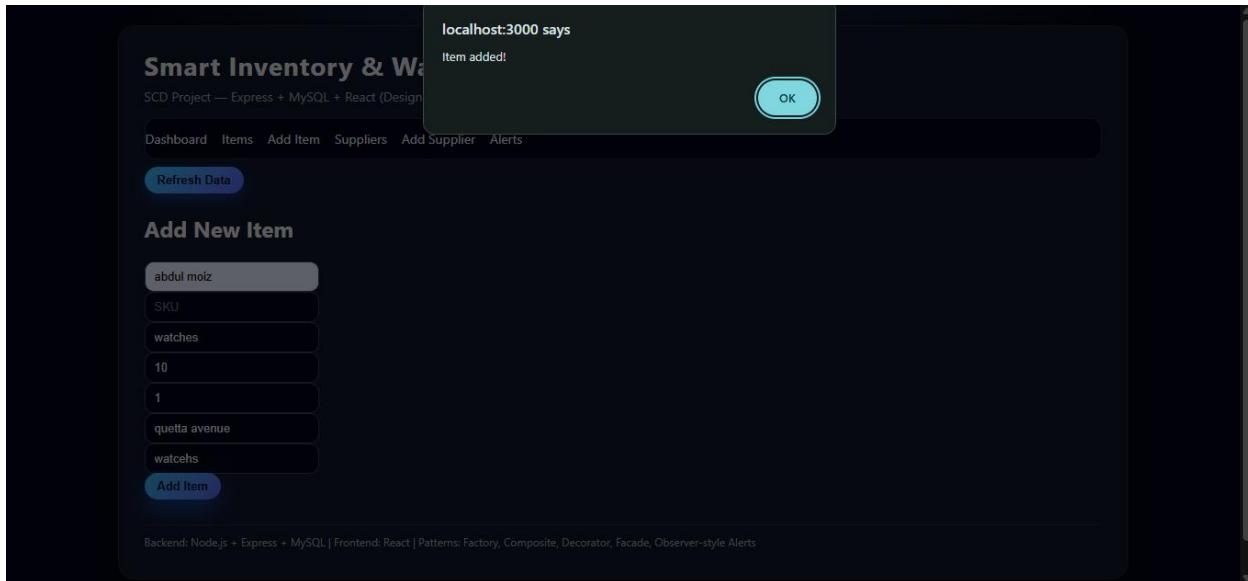
Refresh

Name	SKU	Category	Quantity	Min Stock	Warehouse	Labels
Keyboard	KB-001	Electronics	2	5	A1-B2	NEW
Alert Test	SKU-TEST-22	Test	1	5	Z2	CRITICAL
Dell Laptop	SKU-100	Electronics	15	3	A1-R2	HEAVY
Test Item 1	SKU-001	Test Category	5	2	W1-S1-B1	FRAGILE

Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts

2. Items list

3. Add Item form



4. Suppliers page

The screenshot shows the 'Suppliers List' page of the Smart Inventory & Warehouse System. The page title is 'Smart Inventory & Warehouse System'. It displays a table with two rows of supplier information:

Supplier Name	Contact Person	Email	Phone
Tech Supplies Co	Ahmad Khan	techsupplies@example.com	0300-1234567
Alpha Traders	Bilal Ahmed	alpha@example.com	0312-9988776

The footer contains the text 'Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts'.

Smart Inventory & Warehouse System

SCD Project — Express + MySQL + React (Design Patterns)

Dashboard Items Add Item Suppliers Add Supplier Alerts

Refresh Data

Add Supplier

Name
Contact Person
Email
Phone

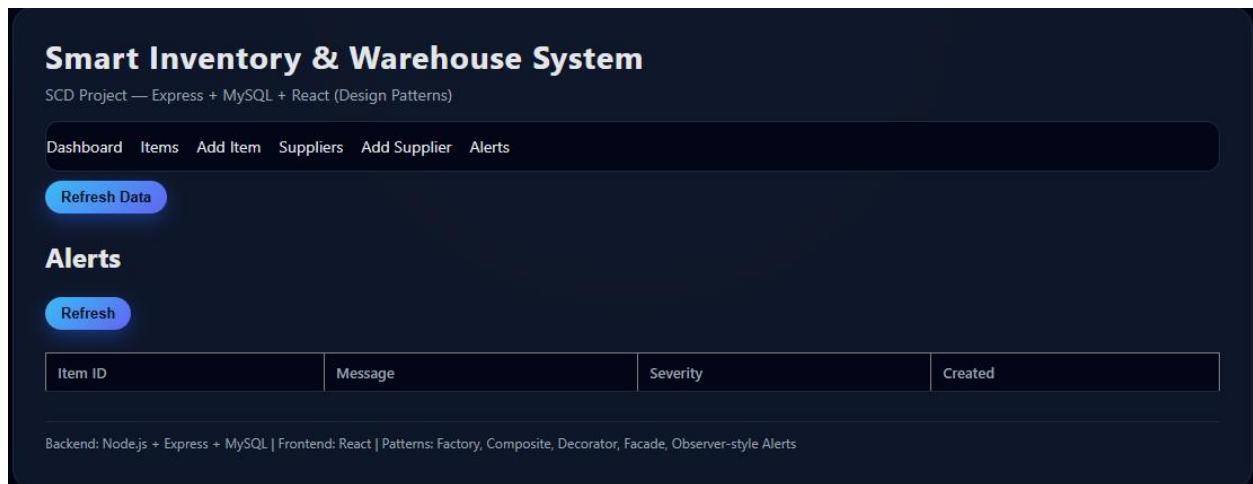
Add Supplier

Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts

The screenshot shows a dark-themed web application interface. At the top, it displays the title 'Smart Inventory & Warehouse System' and a subtitle 'SCD Project — Express + MySQL + React (Design Patterns)'. Below the title is a navigation bar with links: Dashboard, Items, Add Item, Suppliers, Add Supplier, and Alerts. A 'Refresh Data' button is located just below the navigation bar. The main content area is titled 'Add Supplier' and contains four input fields: Name, Contact Person, Email, and Phone. A blue 'Add Supplier' button is positioned at the bottom of the form. At the very bottom of the page, there is a footer note: 'Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts'.

5. Add supplier form

6. Alerts page



The screenshot shows a dark-themed web application interface for a 'Smart Inventory & Warehouse System'. At the top, it says 'SCD Project — Express + MySQL + React (Design Patterns)'. Below that is a navigation bar with links: Dashboard, Items, Add Item, Suppliers, Add Supplier, and Alerts. A blue button labeled 'Refresh Data' is visible. The main section is titled 'Alerts' with a 'Refresh' button. A table header is shown with columns: Item ID, Message, Severity, and Created. At the bottom, there's a footer note: 'Backend: Node.js + Express + MySQL | Frontend: React | Patterns: Factory, Composite, Decorator, Facade, Observer-style Alerts'.

Problems Faced & Solutions

Problem 1: MySQL connection errors & environment variables

Solution: Used .env + proper MySQL credentials + error handling.

Problem 2: Auto-refresh alerts not triggering

Solution: Implemented Observer pattern and database triggers within itemService.

Problem 3: Hard to build warehouse structure

Solution: Composite pattern solved hierarchical structure cleanly.

Problem 4: Repeated logic for item creation

Solution: Implemented Factory pattern for clean object creation.

Problem 5: Dashboard required multiple SQL joins

Solution: Added Reporting Facade for combined results.

Conclusion

This project successfully demonstrates how **software design patterns** improve system structure, maintainability, and scalability.

By using

Factory, Composite, Decorator, Observer, and Facade Patterns,
the system becomes modular and easy to extend.

The backend (Node + Express + MySQL) and frontend (React.js) work together to provide a complete Smart Inventory Management solution.

This project fulfills all SCD course requirements with proper pattern implementations, clean UI, and professional system architecture.